Compound Markets Governance Docs Search the docs

Compound v2 cTokens Comptroller Governance Open Price Feed API Security

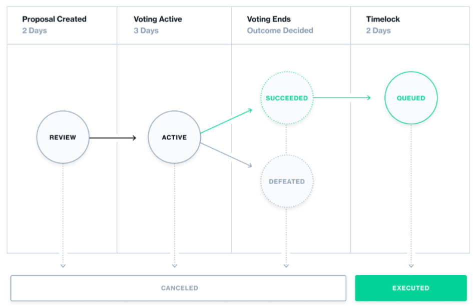
Governance

Introduction

The Compound protocol is governed and upgraded by COMP token-holders, using three distinct components; the COMP token, governance module (Governor Bravo), and Timelock. Together, these contracts allow the community to propose, vote, and implement changes through the administrative functions of a cToken or the Comptroller. Proposals can modify system parameters, support new markets, or add entirely new functionality to the protocol.

COMP token-holders can delegate their voting rights to themselves, or an address of their choice. Addresses delegated at least 25,000 COMP can create governance proposals; any address can lock 100 COMP to create an Autonomous Proposal, which becomes a governance proposal after being delegated 25,000 COMP.

When a governance proposal is created, it enters a 2 day review period, after which voting weights are recorded and voting begins. Voting lasts for 3 days; if a majority, and at least 400,000 votes are cast for the proposal, it is queued in the Timelock, and can be implemented 2 days later. In total, any change to the protocol takes at least one week.



COMP

COMP is an ERC-20 token that allows the owner to delegate voting rights to any address, including their own address. Changes to the owner's token balance automatically adjust the voting rights of the delegate.

Delegate

Delegate votes from the sender to the delegatee. Users can delegate to 1 address at a time, and the number of votes added to the delegatee's vote count is equivalent to the balance of COMP in the user's account. Votes are delegated from the current block and onward, until the sender delegates again, or transfers their COMP.

COMP

```
function delegate(address delegatee)
```

- `delegatee`: The address in which the sender wishes to delegate their votes to.
- `msg.sender`: The address of the COMP token holder that is attempting to delegate their votes.
- `RETURN`: No return, reverts on error.

Solidity

```
Comp comp = Comp(0x123...); // contract address
comp.delegate(delegateeAddress);
```

Web3 1.2.6

```
const tx = await comp.methods.delegate(delegateeAddress).send({ from: sender });
```

Delegate By Signature

Delegate votes from the signatory to the delegatee. This method has the same purpose as Delegate but it instead enables offline signatures to participate in Compound governance vote delegation. For more details on how to create an offline signature, review EIP-712.

COMP

```
function delegateBySig(address delegatee, uint nonce, uint expiry, uint8 v, bytes32 r, bytes32 s)
```

- `delegatee`: The address in which the sender wishes to delegate their votes to.
- `nonce`: The contract state required to match the signature. This can be retrieved from the contract's public nonces mapping.
- `expiry`: The time at which to expire the signature. A block timestamp as seconds since the unix epoch (uint).
- `v`: The recovery byte of the signature.
- `r`: Half of the ECDSA signature pair.
- `s`: Half of the ECDSA signature pair.
- `RETURN`: No return, reverts on error.

Solidity

```
Comp comp = Comp(0x123...); // contract address
comp.delegateBySig(delegateeAddress, nonce, expiry, v, r, s);
```

Web3 1.2.6

```
const tx = await comp.methods.delegateBySig(delegateeAddress, nonce, expiry, v, r, s).send({});
```

Get Current Votes

Gets the balance of votes for an account as of the current block.

COMP

```
function getCurrentVotes(address account) returns (uint96)
```

- `account`: Address of the account in which to retrieve the number of votes.
- `RETURN`: The number of votes (integer).

Solidity

```
Comp comp = Comp(0x123...); // contract address
uint votes = comp.getCurrentVotes(0xabc...);
```

Web3 1.2.6

```
const account = '0x123...'; // contract address
const votes = await comp.methods.getCurrentVotes(account).call();
```

Get Prior Votes

Gets the prior number of votes for an account at a specific block number. The block number passed must be a finalized block or the function will revert.

COMP

```
function getPriorVotes(address account, uint blockNumber) returns (uint96)
```

- `account`: Address of the account in which to retrieve the prior number of votes.
- `blockNumber`: The block number at which to retrieve the prior number of votes.
- `RETURN`: The number of prior votes.

Solidity

```
Comp comp = Comp(0x123...); // contract address
uint priorVotes = comp.getPriorVotes(account, blockNumber);
```

Web3 1.2.6

```
const priorVotes = await comp.methods.getPriorVotes(account, blockNumber).call();
```

Key Events

Event	Description
`DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate)`	An event thats emitted when an account changes its delegate.
`DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance)`	An event thats emitted when a delegate account's vote balance changes.
`ProposalCreated(uint id, address proposer, address[] targets, uint[] values, string[] signatures, bytes[] calldatas, uint startBlock, uint endBlock, string description)`	An event emitted when a new proposal is created.
`VoteCast(address voter, uint proposalId, bool support, uint votes)`	An event emitted when a vote has been cast on a proposal.
`ProposalCanceled(uint id)`	An event emitted when a proposal has been canceled.
`ProposalQueued(uint id, uint eta)`	An event emitted when a proposal has been queued in the Timelock.
`ProposalExecuted(uint id)`	An event emitted when a proposal has been executed in the Timelock.

Governor Bravo

Governor Bravo is the governance module of the protocol; it allows addresses with more than 25,000 COMP to propose changes to the protocol. Addresses that held voting weight, at the start of the proposal, invoked through the getpriorvotes function, can submit their votes during a 3 day voting period. If a majority, and at least 400,000 votes are cast for the proposal, it is queued in the Timelock, and can be implemented after 2 days.

Quorum Votes

The required minimum number of votes in support of a proposal for it to succeed.

Governor Bravo

```
function quorumVotes() public pure returns (uint)
```

- `RETURN`: The minimum number of votes required for a proposal to succeed.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
uint quorum = gov.quorumVotes();
```

Web3 1.2.6

```
const quorum = await gov.methods.quorumVotes().call();
```

Proposal Threshold

The minimum number of votes required for an account to create a proposal. This can be changed through governance.

Governor Bravo

```
function proposalThreshold() returns (uint)
```

- `RETURN`: The minimum number of votes required for an account to create a proposal.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
uint threshold = gov.proposalThreshold();
```

Web3 1.2.6

```
const threshold = await gov.methods.proposalThreshold().call();
```

Proposal Max Operations

The maximum number of actions that can be included in a proposal. Actions are functions calls that will be made when a proposal succeeds and executes.

Governor Bravo

```
function proposalMaxOperations() returns (uint)
```

- `RETURN`: The maximum number of actions that can be included in a proposal.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
uint operations = gov.proposalMaxOperations();
```

Web3 1.2.6

```
const operations = await gov.methods.proposalMaxOperations().call();
```

Voting Delay

The number of Ethereum blocks to wait before voting on a proposal may begin. This value is added to the current block number when a proposal is created. This can be changed through governance.

Governor Bravo

```
function votingDelay() returns (uint)
```

- `RETURN`: Number of blocks to wait before voting on a proposal may begin.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
uint blocks = gov.votingDelay();
```

Web3 1.2.6

```
const blocks = await gov.methods.votingDelay().call();
```

Voting Period

The duration of voting on a proposal, in Ethereum blocks. This can be changed through governance.

Governor Bravo

```
function votingPeriod() returns (uint)
```

- `RETURN`: The duration of voting on a proposal, in Ethereum blocks.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
uint blocks = gov.votingPeriod();
```

Web3 1.2.6

```
const blocks = await gov.methods.votingPeriod().call();
```

Propose

Create a Proposal to change the protocol. E.g., A proposal can set a cToken's interest rate model or risk parameters on the Comptroller. Proposals will be voted on by delegated voters. If there is sufficient support before the voting period ends, the proposal shall be automatically enacted. Enacted proposals are queued and executed in the Compound Timelock contract.

The sender must hold more COMP than the current proposal threshold (`proposalThreshold()`) as of the immediately previous block. If the threshold is 25,000 COMP, the sender must have been delegated more than 1% of all COMP in order to create a proposal. The proposal can have up to 10 actions (based on `proposalMaxOperations()`).

The proposer cannot create another proposal if they currently have a pending or active proposal. It is not possible to queue two identical actions in the same block (due to a restriction in the Timelock), therefore actions in a single proposal must be unique, and unique proposals that share an identical action must be queued in different blocks.

Governor Bravo

```
function propose(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[]
```

- `targets`: The ordered list of target addresses for calls to be made during proposal execution. This array must be the same length as all other array parameters in this function.
- `values`: The ordered list of values (i.e. msg.value) to be passed to the calls made during proposal execution. This array must be the same length as all other array parameters in this function.
- `signatures`: The ordered list of function signatures to be passed during execution. This array must be the same length as all other array parameters in this function.
- `calldatas`: The ordered list of data to be passed to each individual function call during proposal execution. This array must be the same length as all other array parameters in this function.
- `description`: A human readable description of the proposal and the changes it will enact.
- `RETURN`: The ID of the newly created proposal.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
uint proposalId = gov.propose(targets, values, signatures, calldatas, description);
```

Web3 1.2.6

```
const tx = gov.methods.propose(targets, values, signatures, calldatas, description).send({ from: sen
```

Queue

After a proposal has succeeded, it is moved into the Timelock waiting period using this function. The waiting period (e.g. 2 days) begins when this function is called. The queue function can be called by any Ethereum address.

Governor Bravo

```
function queue(uint proposalId)
```

- `proposalId`: ID of a proposal that has succeeded.
- `RETURN`: No return, reverts on error.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
gov.queue(proposalId);
```

Web3 1.2.6

```
const tx = gov.methods.queue(proposalId).send({ from: sender });
```

Execute

After the Timelock waiting period has elapsed, a proposal can be executed using this function, which applies the proposal changes to the target contracts. This will invoke each of the actions described in the proposal. The execute function can be called by any Ethereum address. Note: this function is *payable*, so the Timelock contract can invoke payable functions that were selected in the proposal.

Governor Bravo

```
function execute(uint proposalId) payable
```

- `proposalId`: ID of a succeeded proposal to execute.
- `RETURN`: No return, reverts on error.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
gov.execute(proposalId).value(999).gas(999)();
```

Web3 1.2.6

```
const tx = gov.methods.execute(proposalId).send({ from: sender, value: 1 });
```

Cancel

A proposal is eligible to be cancelled at any time prior to its execution, including while queued in the Timelock, using this function.

The cancel function can be called by the proposal creator, or any Ethereum address, if the proposal creator fails to maintain more delegated votes than the proposal threshold (e.g. 25,000).

Governor Bravo

```
function cancel(uint proposalId)
```

- `proposalId`: ID of a proposal to cancel. The proposal cannot have already been executed.
- `RETURN`: No return, reverts on error.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
gov.cancel(proposalId);
```

Web3 1.2.6

```
const tx = gov.methods.cancel(proposalId).send({ from: sender });
```

Get Actions

Gets the actions of a selected proposal. Pass a proposal ID and get the targets, values, signatures and calldatas of that proposal.

Governor Bravo

```
function getActions(uint proposalId) returns (uint proposalId) public view returns (address[] memory
```

- `proposalId`: ID of a proposal in which to get its actions.
- `RETURN`: Reverts if the proposal ID is invalid. If successful, the following 4 references are returned.
  - Array of addresses of contracts the proposal calls.
  - Array of unsigned integers the proposal uses as values.
  - Array of strings of the proposal's signatures.
  - Array of calldata bytes of the proposal.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
uint proposalId = 123;
(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldata
```

Web3 1.2.6

```
const {0: targets, 1: values, 2: signatures, 3: calldatas} = gov.methods.getActions(proposalId).call
```

Get Receipt

Gets a proposal ballot receipt of the indicated voter.

Governor Bravo

```
function getReceipt(uint proposalId, address voter) returns (Receipt memory)
```

- `proposalId`: ID of the proposal in which to get a voter's ballot receipt.
- `voter`: Address of the account of a proposal voter.
- `RETURN`: Reverts on error. If successful, returns a Receipt struct for the ballot of the voter address.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
Receipt ballot = gov.getReceipt(proposalId, voterAddress);
```

Web3 1.2.6

```
const proposalId = 11;
const voterAddress = '0x123...';
const result = await gov.methods.getReceipt(proposalId, voterAddress).call();
const { hasVoted, support, votes } = result;
```

State

Gets the proposal state for the specified proposal. The return value, `ProposalState` is an enumerated type defined in the Governor Bravo contract.

Governor Bravo

```
function state(uint proposalId) returns (ProposalState)
```

- `proposalId`: ID of a proposal in which to get its state.
- `RETURN`: Enumerated type ProposalState. The types are Pending, Active, Canceled, Defeated, Succeeded, Queued, Expired, andExecuted.

Solidity

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
GovernorBravo.ProposalState state = gov.state(123);
```

Web3 1.2.6

```
const proposalStates = ['Pending', 'Active', 'Canceled', 'Defeated', 'Succeeded', 'Queued', 'Expired
const proposalId = 123;
result = await gov.methods.state(proposalId).call();
const proposalState = proposalStates[result];
```

Cast Vote

Cast a vote on a proposal. The account's voting weight is determined by the number of votes the account had delegated to it at the time the proposal state became active.

Governor Bravo

```
function castVote(uint proposalId, uint8 support)
```

- `proposalId`: ID of a proposal in which to cast a vote.
- `support`: An integer of 0 for against, 1 for in-favor, and 2 for abstain.
- `RETURN`: No return, reverts on error.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
gov.castVote(proposalId, 1);
```

Web3 1.2.6

```
const tx = gov.methods.castVote(proposalId, 0).send({ from: sender });
```

Cast Vote With Reason

Cast a vote on a proposal with a reason attached to the vote.

Governor Bravo

```
function castVoteWithReason(uint proposalId, uint8 support, string calldata reason)
```

- `proposalId`: ID of a proposal in which to cast a vote.
- `support`: An integer of 0 for against, 1 for in-favor, and 2 for abstain.
- `reason`: A string containing the voter's reason for their vote selection.
- `RETURN`: No return, reverts on error.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
gov.castVoteWithReason(proposalId, 2, "I think...");
```

Web3 1.2.6

```
const tx = gov.methods.castVoteWithReason(proposalId, 0, "I think...").send({ from: sender });
```

Cast Vote By Signature

Cast a vote on a proposal. The account's voting weight is determined by the number of votes the account had delegated at the time that proposal state became active. This method has the same purpose as Cast Vote but it instead enables offline signatures to participate in Compound governance voting. For more details on how to create an offline signature, review EIP-712.

Governor Bravo

```
function castVoteBySig(uint proposalId, uint8 support, uint8 v, bytes32 r, bytes32 s)
```

- `proposalId`: ID of a proposal in which to cast a vote.
- `support`: An integer of 0 for against, 1 for in-favor, and 2 for abstain.
- `v`: The recovery byte of the signature.
- `r`: Half of the ECDSA signature pair.
- `s`: Half of the ECDSA signature pair.
- `RETURN`: No return, reverts on error.

Solidity

```
GovernorBravo gov = GovernorBravo(0x123...); // contract address
gov.castVoteBySig(proposalId, 0, v, r, s);
```

Web3 1.2.6

```
const tx = await gov.methods.castVoteBySig(proposalId, 1, v, r, s).send({});
```

Timelock

Each protocol contract is controlled by the Timelock contract, which can modify system parameters, logic, and contracts in a 'time-delayed, opt-out' upgrade pattern. The Timelock has a hard-coded minimum delay of 2 days, which is the least amount of notice possible for a governance action. The Timelock contract queues and executes proposals that have passed a Governance vote.

Pause Guardian

The Comptroller contract designates a Pause Guardian address capable of disabling protocol functionality. Used only in the event of an unforeseen vulnerability, the Pause Guardian has one and only one ability: to disable a select set of functions: Mint, Borrow, Transfer, and Liquidate. The Pause Guardian cannot unpause an action, nor can it ever prevent users from calling Redeem, or Repay Borrow to close positions and exit the protocol. COMP token-holders designate the Pause Guardian address, which is held by the Community Multi-Sig.